Submitted to VLSI Design

# Writing Research Software in a Large Group for the NEMO Project

Gerhard Klimeck, Dan Blanks, Roger Lake, R. Chris Bowen, Chenjing L. Fernando, Manhua Leng, William R. Frensley*, Dejan Jovanovic, and Paul Sotirelis

Corporate R&D, Texas Instruments Incorporated, Dallas, TX 75265
*School of Engineering, The University of Texas at Dallas, Richardson, TX 75083

The nanoelectronic modeling (NEMO) program is the result of a three-year development effort involving four universities and the Corporate Research & Development laboratory of Texas Instruments to create a comprehensive quantum device modeling tool for layered semiconductor structures. Based on the non-equilibrium Green function formalism, it includes the effects of quantum charging, bandstructure and incoherent scattering from alloy disorder, interface roughness, acoustic phonons, and polar optical phonons. NEMO addresses the diverse needs of two different types of users: (i) the engineer/experimentalist who desires a black-box design tool and (ii) the theorist who is interested in a detailed investigation of the physics. A collection of models trade off physical content with speed and memory requirements. Access to this comprehensive theoretical framework is accommodated by a graphical user interface (GUI) that facilitates device prototyping and *in-situ* data analysis. We describe a hierarchical software design that allows rapid incorporation of theory enhancements while maintaining a user-friendly GUI, thus satisfying the conflicting criteria of ease of use and ease of development. The theory and GUI modules share data structures that define the device structure, material parameters, and simulation parameters. These data structures may contain general data such as integer and real numbers, option lists, vectors, matrices and the labels for both batch and GUI operation. NEMO generates the corresponding GUI elements at run-time for display and entry of these data structures.

## Introduction

At the inception of NEMO in 1993, quantum transport programs were in their infancy as compared to the physical comprehensiveness of semiclassical Monte Carlo simulators. The existing software only addressed a limited range of quantum devices and quantum transport theory. The NEMO project was conceived to create a single software package that would simultaneously include bandstructure effects, self-consistent charging effects, and incoherent scattering effects, and be flexible enough to model a wide variety of device designs.

Texas Instruments and the collaborating universities evaluated a number of different formalisms (see Fig. 1) and chose the non-equilibrium Green function (NEGF) approach as the most general and flexible alternative. The multiple sequential scattering algorithm was incorporated within the NEGF formalism and the other approaches were eventually dropped. The theory used to generate NEMO calculations is presented in other publications[1,2].

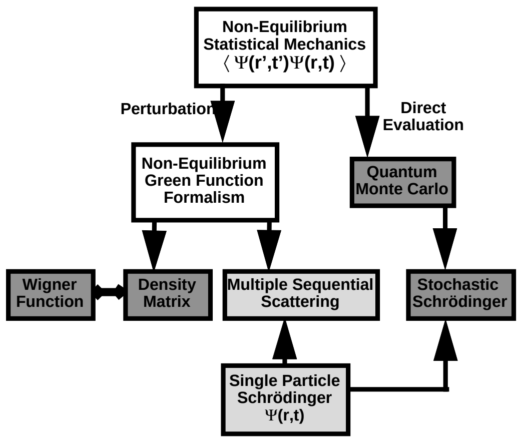

Non-Equilibrium Statistical Mechanics
$\langle \Psi(r',t')\Psi(r,t) \rangle$
Perturbation
Direct Evaluation
Non-Equilibrium Green Function Formalism
Quantum Monte Carlo
Wigner Function
Density Matrix
Multiple Sequential Scattering
Stochastic Schrödinger
Single Particle Schrödinger
$\Psi(r,t)$

FIG. 1. Quantum transport theories pursued during the NEMO project. The light gray background indicates the incorporation into the NEGF approach. The dark background indicates approaches that were dropped.

This paper addresses the software engineering aspects of writing NEMO in terms of the basic software design and some of the most important software methods we used to facilitate the tasks of both the programmer and the user. The first section reviews the fundamental program design. This is followed by a discussion of the event-driven methodology used to construct the simulation parameters user interface. The next section reviews the method used to transfer data between NEMO data structures, files, and the graphical user interface. The concluding section summarizes the development environment and software tools used to construct NEMO.

## Fundamental Program Design

The basic content of the NEMO software is shown in Fig. (2). At the heart lies the NEGF formalism and open system boundary conditions. The next shell lists the physical models which may be included within several different approximations. The outer shell lists the software algorithms and features that support the interior calculation code.

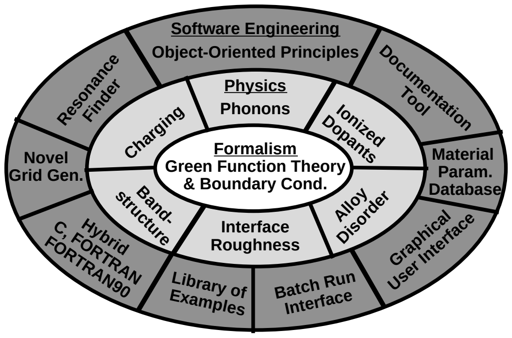


FIG. 2. Content of the NEMO software.

## Batch vs. GUI Program Design

In its original form, NEMO was a command-line program. This "NEMO-Batch" version was later augmented with a graphical user interface (GUI) that provides graphical methods for entry of input parameters, plots calculation results both during and after the calculation, and allows the user to compare data and calculations all in the same program. Unlike NEMO-Batch, which performs the simulation without interruption until completion, the NEMO-GUI program allows the user to start, interrupt, and abort calculations at any time.

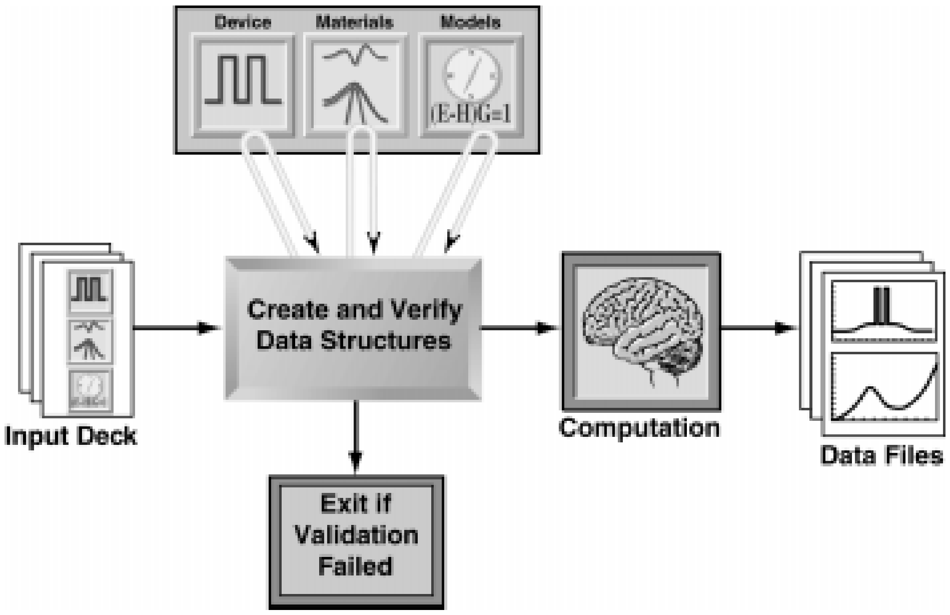
Despite the obvious advantages of the GUI interface, NEMO-Batch was still very useful. Since it excludes all GUI-related functionality, NEMO-Batch uses less memory, runs faster, and is easier to port to different computer platforms than NEMO-GUI. Consequently, we devised a method to retain the NEMO-Batch code as the heart of the NEMO-GUI program. To completely understand how NEMO operates, we must examine the design of both the batch and GUI versions of NEMO.

Figure 3(a) illustrates the fundamental operation of NEMO-Batch. The Input Deck is a text file that contains the input parameter required for the calculation. The user calls NEMO-Batch on a command line using the filename of the Input Deck as the sole argument. NEMO-Batch reads and processes the Input Deck, runs the computation, stores the results in output data files, and then exits. A NEMO-Batch run can perform the calculation for a single bias point or a sequence of bias points. While a current vs. voltage calculation is a standard output, many other outputs are available, including charge density, band profile, and transmission coefficients.

As shown in Fig. 3(b), the NEMO-GUI operation includes most of the NEMO-Batch code. This design requires a separation of the GUI and batch code into different modules. The NEMO-GUI compilation includes all of the program modules whether they are batch or GUI-related. The NEMO-Batch compilation excludes purely GUI modules from the executable. There are some cases where a mixture of batch and GUI code is unavoidable, particularly in message routines and the functions that plot and store calculation results during the calculation. For those situations, a preprocessor-defined flag effectively deletes GUI-related code from any routines with a mix of batch-GUI code.

Both NEMO-Batch and NEMO-GUI use the same verification procedures to check the validity of the input parameters. The validation routine determines the parameters required for the requested calculation and checks for their presence in the input deck. If any material or model parameters are missing, NEMO provides default values from its internal database. In this manner, users can either accept the default parameters or input their own parameters. This feature addresses the needs of the device physicist who wants detailed control over the calculation process and the device engineer who only wants to determine general device behavior as quickly as possible.

(a)



(b)

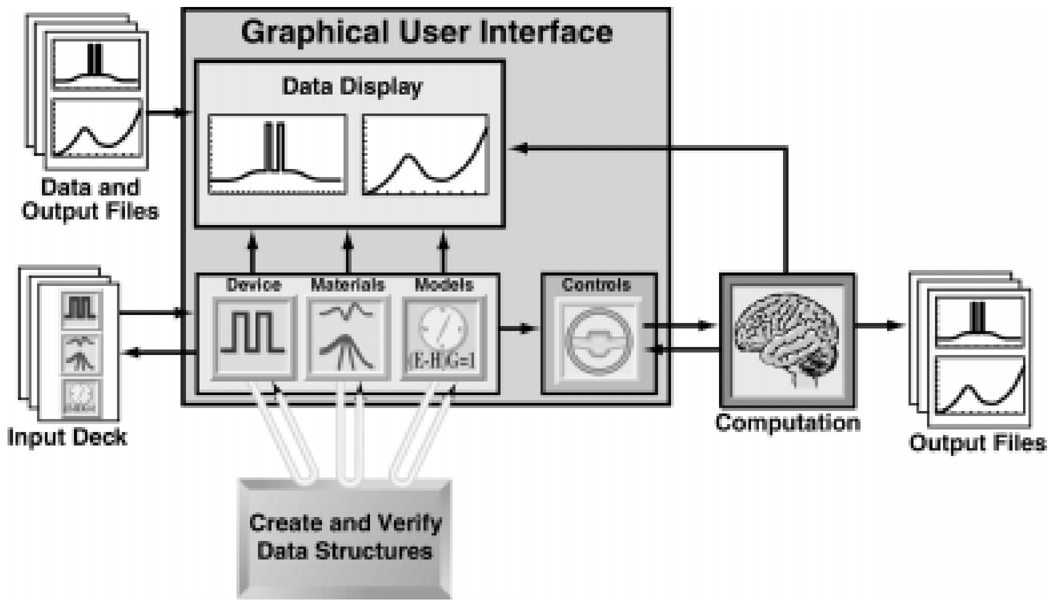


FIG. 3. Basic design of NEMO program for (a) batch and (b) GUI operation. Input parameters are classified as device structure parameters, material parameters, and model parameters. Device structure parameters specify the device geometry, layer composition, and any other external physical parameters such as the terminal connections and temperature. Material parameters include the band structure information such as effective mass, energy gaps, and band offsets. Model parameters specify the theoretical models used for the calculation such as band structure, charge self-consistency, scattering, simulation domains, and grids.

Even though they use the same validation routines, NEMO-Batch and NEMO-GUI have very different means of responding to validation errors. For NEMO-Batch, the validation routine only needs to check parameters read from the Input Deck. If an error occurs in the verification procedure, NEMO-Batch must exit and report the error to a log file.

NEMO-GUI not only checks the Input Deck values, but also validates parameters as the user enters them into the GUI. If an error occurs for either case, NEMO-GUI displays an error message with advice on why the error occurred and how to remedy the problem. The user must input the correct information before starting the calculation or saving the input parameters. This procedure ensures that an Input Deck created using NEMO-GUI will be valid for NEMO-Batch operation. Consequently, even when using NEMO-Batch for the calculation, NEMO-GUI provides the most convenient method to construct the input deck.

## Simulation Parameter Input Methodology

NEMO incorporates a wide spectrum of models with varying tradeoffs between accuracy and speed. This range of models is essential given the large range of devices we wish to model for different scientific and engineering applications.

The downside of this flexibility is that NEMO has over 100 simulation parameters, many of which interact in complex ways. In traditional simulators, the user must comprehend all of the simulation parameters, carefully select the essential parameters for a particular study, and ensure that all parameter values are self-consistent. This can be a daunting task for even the simplest simulation.

It is equally difficult for the programmer to deal with for two reasons: (1) New models and parameters added to the simulator must be reflected in the GUI as quickly as possible, and (2) simulation parameters should only be defined in a single module accessed by both the theory and the GUI to ease the code maintenance.

Our approach to this problem was to create the GUI interface at run-time based on a hierarchical system of simulation parameters. These simulation parameters are defined in a single module shared by the GUI and theory code. A key advantage to this method is that it insulates changes in the simulation parameter set from changes in the GUI and vice versa. The next section reviews the method used to create the run-time GUI interface.

The most basic level of the hierarchy directs the choice of fundamental models such as the potential, band structure, and scattering models. NEMO determines the data structures required by these models and creates a new GUI interface. The GUI displays default choices that the user can either accept or change.

When the user prompts NEMO to accept these parameters, NEMO tests for valid user input and enforces consistency between interacting parameters. For example, if one parameter value must be larger than another parameter, NEMO forces adherence to this rule.

After validating the selected options, NEMO determines the data structures required by the next level on the hierarchy, creates and displays a new GUI interface, and repeats the process until the parameter set is sufficient to run the simulation. If the user goes back to any portion of the hierarchy to select new options, parameters at higher levels of the hierarchy remain intact, while parameters at the lower levels are added and subtracted as appropriate.

This event-driven procedure only presents the parameters needed by the selected models. All other parameters are either hidden or are disabled from user entry. Limiting the parameter display in this manner greatly reduces the sheer number of simulation parameters presented to the user. Since NEMO provides default values for all simulation parameters, the user can often specify a few of the top-level parameters and then prompt NEMO to use the default values for the remainder of the hierarchy. In most cases, this approach generates a reasonable simulation. Nevertheless, the user has the option of adjusting all parameters down to the lowest level of the hierarchy, thus fulfilling a primary goal of NEMO to give the user full control over all aspects of the simulation.

## Data Transfer and Run-Time GUI Creation

The NEMO program must transfer large amounts of information between data structures, files, and the GUI. For a small program, it is sufficient to write customized routines for these operations, but this approach became impractical as NEMO grew in size. To resolve this problem, we designed a generic algorithm for transferring. information to and from data structures. In addition to transferring information, this same algorithm creates portions of the GUI at run-time for parameter entry. A detailed description of this algorithm is beyond the scope of this paper, but due to its importance to NEMO and its utility to both the user and the programmer, we will highlight the basic features.

We refer to this process as the MemberDescriptor algorithm. We defined a MemberDescriptor data structure that contains all of the information needed to translate values between a data structure member and its destination. The MemberDescriptor includes a character string identifier, the memory location of a data structure member relative to the data structure address, and an enumerated flag that defines the data type (number, character string, option list, etc.). Each data structure has at least one associated MemberDescriptor array. Each element of the MemberDescriptor array corresponds to one of the data structure members. To transfer information from the data structure, the translation routine uses the base address of the data structure and the MemberDescriptor array to derive the values of each data structure member. The inverse of this procedure transfers values from either the GUI or a file into each data structure member.

For example, assume we wish to transfer the value of an integer data structure member to a file. The translation routine adds the relative position of the integer data structure to the base address of the data structure. To determine the actual integer value, the translator casts the contents of the memory location to an integer. The derived integer value is stored in the file in the general format:

```
<identifier> = <value>
```

where the `<identifier>` is the character string identifier set in the MemberDescriptor data structure and `<value>` is the integer number. Reversing this process inputs the same information from the file and stores the integer value into the data structure.

This procedure is used in many programs as a generic method for file input and output[3]. For NEMO, we extended the algorithm to transfer information between the GUI and the internal data structures. Figure 4 illustrates how data values are transferred between data structures and a GUI panel consisting of an option list, a text box, a check box, and a graphics plot with adjustable cursors.

For the simulation parameter input method discussed in the last section, we use the MemberDescriptor algorithm to create the GUI interface for each hierarchy. This dynamic design vastly enhances the usability of the GUI compared to a traditional static design that would require a fixed set of GUI elements throughout the program operation.

The MemberDescriptor approach saves a great deal of time and trouble for the programmer. Adding a new data structure no longer requires customized code to transfer values between data structures, files, and the GUI. Only a new MemberDescriptor array is needed. In fact, even software team members who are totally ignorant of GUI programming techniques can effectively "program" the GUI in this manner.

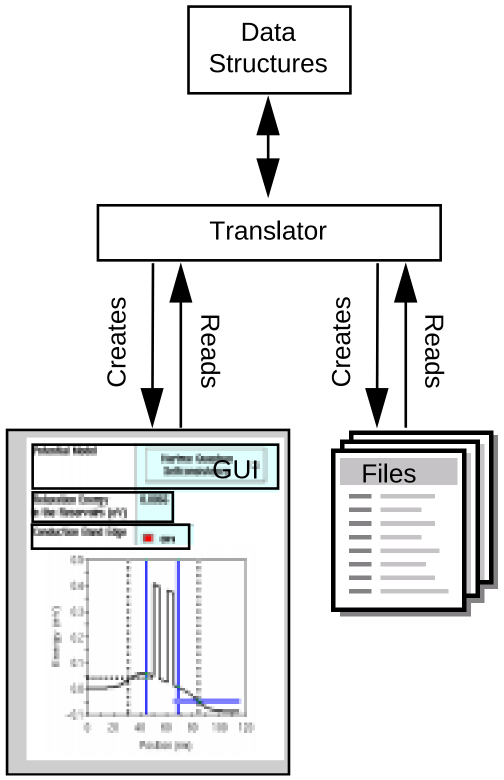


FIG. 4. Generic data transfer between NEMO data structures, the graphical user interface, and files.

## Development Environment

The following summarizes key elements of the environment used to create the NEMO simulator:

*Programming Language*. Most of the program was coded using ANSI C with some FORTRAN 77 and FORTRAN 90 code used in modules that required the greatest speed.

*Development Platforms*. Code development occurred on Unix workstations either using HP Apollo or SGI platforms. We also ported NEMO to the Sun, IBM, and DEC Unix platforms.

*Program Library System*. To facilitate code sharing and revision updates, we utilized a standard program library system called RCS (Revision Control System). The RCS system manages a check-out and check-in system that only allows one programmer to edit a file at a time. The RCS system also stores changes between code versions and can recover any previous version of the project.

*Graphical User Interface*. The NEMO GUI was written in Motif which uses C-callable routines to create and control GUI objects on a wide variety of Unix platforms.

*GUI Tools*. The static elements of NEMO were designed using the XDesigner software tool. The spreadsheet table and the 2D, 3D, and contour plot tools were programmed using the 3rd party XRT Widget set.

*Run-time Debugger*. NEMO relies heavily on dynamic memory creation techniques that are extremely difficult to debug with conventional debuggers. The Purify debugger provided the means to find memory violations incurred at run-time and greatly simplified the task of tracking down these errors.

## Conclusion

We have presented highlights of the NEMO software project germane to construction of a large-scale device simulator. It is hoped that these methods will prove useful to other groups engaged in similar types of software projects, whether or not they pertain to quantum device simulations.

**References**

[1] P. Roblin and W. Liou, Phys. Rev. B **46**, 2416 (1993).

[2] R. Lake, G. Klimeck, R. C. Bowen, and D. Jovanovic, to appear in J. Appl. Phys. June 1997 (1997).

[3] A. Nye and T. O'Reilly, X Toolkit Intrinsics Programming Manual, OSF/Motif 1.2 edition, Vol. 4 of O'Reilly X-Windows Series, Chapt. 10 (1993).

**Biographies**

Gerhard Klimeck is a Member of the Technical Staff at Texas Instruments Incorporated. His present research interest is the writing the NEMO quantum device simulator.

Dan Blanks is a Member of the Technical Staff at Texas Instruments Incorporated. His present research interest is the design and coding of the graphical user interface of the NEMO program.

Roger Lake is a Member of the Technical Staff at Texas Instruments Incorporated. His present research interest is scattering effects in RTD's.

R. Chris Bowen is a Member of the Technical Staff at Texas Instruments Incorporated. His current research interest is full-band modeling of quantum electron transport in Silicon and III-V structures.

William R. Frensley is a Professor in the Electrical Engineering Dept. at the U. of Texas at Dallas. His current research interest is modeling of quantum electron transport in nanoelectronic devices.